Europäisches Patentamt	European Patent Office	Office européen des brevets

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

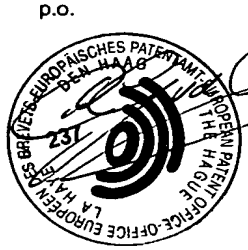
Patentanmeldung Nr. | Patent application No. | Demande de brevet n°

00830673.0

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.



C. v.d. Aa-Jansen

MÜNCHEN, DEN
MUNICH, 21/02/05
MUNICH, LE

Anmeldung Nr:
Application no.: 00830673.0
Demande no:

Anmeldetag:
Date of filing: 17.10.00
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

STMicroelectronics S.r.l.
Via C. Olivetti, 2
20041 Agrate Brianza (Milano)
ITALIE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Processor architecture

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

/00.00.00/

Internationale Patentklassifikation/International Patent Classification/ Classification internationale des brevets:

G06F15/76

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

"Architettura per processori"

* * *

Campo dell'invenzione

La presente invenzione si riferisce alle architetture di processori, in particolare del tipo correntemente denominato "pipeline".

Descrizione della tecnica nota

Uno degli effetti principali dell'introduzione della tecnica di "pipelining" è la modifica della temporizzazione relativa delle istruzioni conseguente alla sovrapposizione parziale della loro esecuzione, il che introduce fattori di conflitto o alea dovuti sia alle dipendenze di dati (data hazard) sia alle modifiche del flusso di controllo (control hazards). In particolare questi conflitti emergono quando l'avvio delle istruzioni attraverso la pipeline modifica l'ordine degli accessi in lettura/scrittura degli operandi rispetto all'ordine naturale del programma (cioè rispetto all'esecuzione sequenziale delle istruzioni in processori non-pipeline).

Al riguardo si può far utilmente riferimento al testo di J. Hennessy and D.A. Patterson, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, San Mateo, CA, Second Edition, 1996.

L'insieme di problemi legati in particolare ai conflitti sui dati può essere risolto a livello hardware con la tecnica di inoltro correntemente denominata "forwarding" (od anche "bypassing" e talvolta "short-circuiting"). Questa tecnica utilizza i registri inter-stadio dell'architettura pipeline per inoltrare i risultati di un'istruzione Ii, prodotti da uno stadio della pipeline, direttamente agli ingressi dei stadi antecedenti della pipeline stessa onde essere utilizzati nell'esecuzione di istruzioni che seguono Ii. Un risultato può essere pertanto inoltrato

dall'uscita di un'unità funzionale verso gli ingressi di un'altra che la precede nel flusso lungo la pipeline, così come a partire dall'uscita di un'unità verso gli ingressi della stessa unità.

Per assicurare questo meccanismo di forwarding è necessario prevedere nel processore i necessari percorsi di inoltro (bypass) e il controllo di tali percorsi. La tecnica di forwarding può richiedere un percorso specifico a partire da ciascun registro della struttura pipeline verso gli ingressi di qualunque unità funzionale, come nel caso dell'architettura denominata DLX alla quale si fa riferimento nel testo già in precedenza citato.

I dati sottoposti a bypass verso le unità funzionali dei primi stadi dell'architettura pipeline vengono di norma comunque memorizzati nel banco di registri (Register File o RF) durante l'ultimo stadio della pipeline (vale a dire il cosiddetto stadio di riscrittura o Write-Back) in vista di un successivo impiego nel programma in via d'esecuzione. I processori che utilizzano la tecnica di forwarding conseguono sostanziali miglioramenti in termini di prestazioni grazie all'eliminazione dei cicli di stallo introdotti da fattori di alea sui dati (data hazard).

I principali problemi legati al meccanismo di forwarding nell'ambito dei processori, ed in particolare nell'ambito dei processori denominati VLIW (acronimo per Very Long Instruction Word) sono stati studiati in lavori quali: A. Abnous and N. Bagherzadeh, "Pipelining and Bypassing in a VLIW Processor", IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 6, June 1994, pp. 658-663 ed H. Corporaal, "Microprocessor Architectures from VLIW to TTA", John Wiley and Sons, England.

In questi lavori sono stati analizzati i vantaggi in termini di prestazioni di vari schemi di bypassing, in particolare per quanto riguarda la loro efficacia nel risolvere alee dati nelle architetture pipeline tanto a quattro quanto a cinque stadi.

Il fatto di sfruttare i valori trasferiti tramite i percorsi di bypass negli stadi della pipeline è stato combinato con l'introduzione di una piccola cache di registri al fine di migliorare le prestazioni, così come descritto nel lavoro di R. Yung and N. C. Wilhelm, "Caching Processor General Registers", ICCD '95, Proceedings of IEEE International Conference on Computer Design, 1995, pp. 307-312. In questa architettura, denominata Register Scoreboard and Cache, gli operandi della pipeline vengono forniti o dalla cache di registri o dalla rete di bypass.

Nel lavoro di L. A. Lozano and G. R. Gao, "Exploiting Short-Lived Variables in Superscalar Processors", MICRO-28, Proceedings of 28th Annual IEEE/ACM International Symposium on Microarchitecture, 1995, pp. 292-302 è stato proposto, in relazione a processori superscalari, uno schema che comprende un'analisi compiuta dal compilatore ed un'estensione dell'architettura al fine di evitare scritture definitive nel banco registri (commit) dei valori di variabili destinate ad avere una breve durata e che di conseguenza non richiedono la persistenza sul lungo periodo nel banco dei registri. I vantaggi forniti da questa soluzione sono stati valutati dagli autori prevalentemente in termini di riduzione delle porte di scrittura verso il banco di registri e di riduzione della quantità di trasferimenti da registri a memoria necessari, così da conseguire miglioramenti nel tempo di esecuzione. Il lavoro citato riporta i miglioramenti legati a questa soluzione in termini di prestazione,

senza che peraltro siano stati considerati gli effetti in termini di assorbimento di potenza.

Il concetto di evitare la presenza nel banco di registri di informazione priva di valore utile (dead value) è stata analizzata nel lavoro di M. M. Martin, A. Roth, C. N. Fischer, "Exploiting Dead Value Information", MICRO-30, Proceedings of 30th Annual IEEE/ACM International Symposium on Microarchitecture, 1997, pp. 125-135. I valori nei registri vengono considerati inutili, ossia "morti", quando essi non vengono letti prima di essere sottoposti a sovrascrittura. I vantaggi di questa soluzione sono stati studiati in termini di riduzione delle dimensioni del banco dei registri e di eliminazione di istruzioni di salvataggio/ripristino non necessarie a partire dal flusso di esecuzione in corrispondenza delle chiamate di procedura e attraverso il cambio di contesto.

Così come dimostrato in lavori quali A. Chandrakasan and R. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits", Proc. of IEEE, 83(4), pp. 498-523, 1995 e K. Roy, S. C. Prasad "Low-Power CMOS VLSI Circuit Design", John Wiley and Sons, Inc., Wiley-Interscience, 2000, un ridotto assorbimento di potenza costituisce un'esigenza sempre più importante per i processori di tipo embedded. Le tecniche a basso assorbimento di potenza vengono ampiamente utilizzate nella progettazione dei microprocessori al fine di far fronte a stringenti limitazioni in termini di assorbimento massimo di potenza e di affidabilità operativa, mantenendo inalterate le caratteristiche in termini di velocità di elaborazione.

La maggior parte delle tecniche a basso assorbimento di potenza sviluppate per i circuiti CMOS digitali mirano a ridurre la potenza di commutazione,

che rappresenta il contributo più significativo al bilancio complessivo di potenza. Per i processori con elevate prestazioni, le soluzioni a basso assorbimento di potenza mirano a ridurre la capacità efficace C_{EFF} dei nodi del processore sottoposti a commutazione. Il parametro C_{EFF} di un nodo è definito come prodotto della capacità del carico C_L e dell'attività di commutazione α del nodo. Nei processori CMOS digitali è possibile ottenere notevoli economie in termini di assorbimento di potenza minimizzando l'attività di transizione dei bus ad alta capacità, come i bus del data-path e i bus di ingresso/uscita. Un'altra parte significativa del bilancio di potenza nei processori moderni è dovuta agli accessi ai banco dei registri multi-porta a ad altri accessi a cache on-chip.

Scopi e sintesi della presente invenzione

La presente invenzione si prefigge lo scopo di fornire un'architettura di processore in grado di superare gli inconvenienti e le limitazioni delineate in precedenza.

Secondo la presente invenzione, tale scopo viene raggiunto grazie ad un'architettura avente le caratteristiche richiamate in modo specifico nelle rivendicazioni che seguono.

In modo specifico, l'obiettivo principale dell'invenzione è quello di definire un criterio di ottimizzazione dell'architettura per processori di tipo pipeline a schedulazione statica ed in particolare per processori pipeline del tipo VLIW in grado di sfruttare la tecnica di forwarding dei dati in relazione alle variabili a vita breve (variabili "short-lived") così da conseguire economie nell'assorbimento di potenza.

In sostanza, l'idea posta alla base dell'invenzione consiste nel ridurre l'attività di accesso al banco di registri (RF), evitando di

memorizzare stabilmente variabili a vita breve. Questo è possibile - con un overhead trascurabile in termini di hardware - grazie alla preesistente disponibilità di registri interstadio e di opportuni percorsi di inoltro (bypass). Le variabili a vita breve vengono semplicemente memorizzate localmente dall'istruzione che le produce nei registri interstadio ed inoltrate direttamente verso l'opportuno stadio dell'istruzione che le utilizza, sfruttando i percorsi di inoltro (bypass). L'istruzione che produce le variabili non attua, quindi, una costosa azione di riscrittura (write-back) verso il banco di registri e l'istruzione che utilizza le variabili non deve realizzare, a sua volta, alcuna lettura a partire dallo stesso banco di registri (RF).

L'applicazione di questa tecnica richiede la valutazione della durata di vita L (liveness length) della n-esima assegnazione ad un registro R, definita come distanza fra la sua n-esima assegnazione ed il suo ultimo impiego. Questa informazione consente di decidere se la variabile deve essere memorizzata nel banco di registri in vista di un uso successivo ovvero se il suo uso è in realtà limitato a pochi cicli di clock. In quest'ultimo caso, la variabile è a vita breve ed il suo valore può essere passato come operando alle istruzioni successive utilizzando i cammini di inoltro (bypass), evitando quindi di scriverlo nel banco di registri.

La decisione relativa al fatto di abilitare la fase di scrittura verso il banco di registri può essere presa dall'hardware in fase di esecuzione oppure anticipata in fase di compilazione del programma sorgente. A differenza di quanto avviene nei processori superscalari, nei quali la maggior parte delle decisioni vengono prese dall'hardware al momento

dell'esecuzione, l'applicazione della tecnica di inoltro (bypass) a basso assorbimento di potenza nelle architetture di tipo VLIW può essere attuata in fase di schedulazione statica da parte del compilatore. Questo modo di procedere riduce la complessità della logica di controllo del processore.

L'architettura proposta diventa particolarmente attraente con riferimento ad alcune applicazioni di tipo embedded, per cui l'analisi della vita utile dei registri ha dimostrato che l'intervallo di riuso di più di metà di tutte le definizioni dei registri è limitata alle due istruzioni successive.

Le caratteristiche più importanti della soluzione secondo l'invenzione sono le seguenti:

- la soluzione secondo l'invenzione propone un'estensione di tipo architetturale della rete di inoltro (bypass) del processore così da evitare la scrittura e la successiva lettura verso il/a partire dal banco di registri delle variabili a vita breve;
- è possibile analizzare gli effetti sul compilatore della soluzione architetturale a basso assorbimento di potenza proposta per i processori di tipo VLIW, con la dimostrazione del fatto che l'implementazione consente di limitare l'overhead in termini di hardware;
- è possibile la trattazione delle eccezioni (esempio: error traps, divisione per zero, ecc.);
- la soluzione secondo l'invenzione è estendibile anche a processori con più di cinque stadi di pipeline (comprendenti più di tre cammini di inoltro) così da ottenere economie in termini di assorbimento di potenza per variabili la cui lunghezza di vita è maggiore di tre;
- la soluzione secondo l'invenzione apre il campo ad ulteriori economie in termini di assorbimento di

potenza ottenibili tramite un'ottimizzazione della schedulazione delle istruzioni, sfruttando al massimo il parallelismo intrinseco di tali processori e mirando a minimizzare la "vita media" (liveness) delle variabili.

Breve descrizione dei disegni annessi

L'invenzione verrà ora descritta, a puro titolo di esempio non limitativo, con riferimento ai disegni annessi, nei quali:

- la figura 1 è un diagramma che illustra i risultati di un'analisi condotta in relazione alla vita attiva dei registri nell'ambito di un processore,
- la figura 2 illustra, sotto forma di uno schema a blocchi funzionale, l'applicazione di un'architettura secondo l'invenzione nell'ambito di un processore,
- la figura 3 è un diagramma che illustra le modalità con cui si procede alla trattazione delle eccezioni nell'ambito di un processore secondo l'invenzione, e
- le figure 4 e 5 illustrano in ancora maggior dettaglio le modalità di attuazione di una soluzione secondo l'invenzione.

Descrizione dei fondamenti teorici dell'invenzione

Come premessa alla descrizione particolareggiata di un esempio di attuazione dell'invenzione, è utile riportare nel seguito i risultati di alcune analisi sperimentali condotte al fine di stabilire la durata di vita (liveness) delle variabili in applicazioni di tipo embedded.

Lo scopo specifico è stato quello di misurare - in fase di esecuzione - la percentuale di definizioni di registri nel codice applicativo che possono essere lette direttamente dalla rete di inoltro, senza essere scritte nel banco di registri RF.

L'analisi sopra descritta è stata condotta utilizzando a titolo di esempio un insieme di algoritmi DSP di impiego corrente scritti in linguaggio C e compilati con un compilatore VLIW industriale su 32 bit e 4 vie.

L'analisi del tempo di vita dei registri può essere condotta sia in modo statico sia in modo dinamico.

L'analisi di tipo statico consiste nell'ispezionare in modo statico il codice assembler generato dal compilatore nell'ambito di ciascun blocco basico così da rilevare la durata del tempo di vita dei registri.

L'analisi di tipo dinamico consiste nell'ispezione delle tracce di esecuzione del codice assembler, che fornisce un'informazione di profiling più accurata in relazione agli accessi ai registri in lettura/scrittura.

I risultati riportati nel seguito si riferiscono alla soluzione dinamica.

Ciascun programma di prova è stato opportunamente strumentato a livello di assembler con un tool automatico e poi simulato, così da tenere conto delle informazioni rilevanti in corrispondenza di ciascun ciclo di clock:

- definizioni dei registri;
- impieghi dei registri; e
- confini dei blocchi basici riscontrati.

Per ciascun blocco basico nella traccia, l'analisi del tempo di vita dei registri è stata realizzata definendo la durata di vita L della n-esima assegnazione ad un registro R come la distanza (misurata in numero di istruzioni) fra la n-esima assegnazione e il suo ultimo utilizzo:

$$L_n(R) = U_n(R) - D_n(R)$$

dove $D_n(R)$ è l'indice di traccia dell'istruzione che ha realizzato la n-esima assegnazione ad R e $U_n(R)$ è l'indice dell'ultima istruzione che ha utilizzato la n-esima assegnazione a R prima della ridefinizione di R durante la (n+1)-esima assegnazione D_{n+1} (R).

In un'architettura VLIW si può assumere un throughput pari ad un'istruzione lunga per ciclo di clock.

Volendo mantenersi in termini di un'analisi molto conservativa, il calcolo di $L_n(R)$ è stato effettuato con le seguenti limitazioni:

- U_n e D_n sono nello stesso blocco basico;
- D_{n+1} e D_n sono nello stesso blocco basico.

Queste regole consentono di semplificare l'analisi considerando unicamente tempi di vita che non attraversino i confini fra i blocchi basici. Tuttavia, questa ipotesi non costituisce limitazione rilevante, dal momento che la maggior parte dei compilatori VLIW moderni massimizzano le dimensioni dei blocchi basici, generando così un numero rilevante di campi di vita che si risolvono completamente all'interno dei rispettivi blocchi basici.

Per chiarire questo concetto, si può analizzare una traccia di codice assembler per una macchina VLIW su quattro vie che esegue un algoritmo DCT (Discrete Cosine Transform). Il codice analizzato è costituito da quattro istruzioni lunghe (qui indicate come 27268, 27269, 27270 e 27271):

```
27268    shr $r16 = $r16, 8
         sub $r18 = $r18, $r7
         add $r17 = $r17, $r19
         sub $r19 = $r19, $r15 ;

27269    shr $r18 = $r18, 8
         shr $r17 = $r17, 8
```

```
                shr $r19 = $r19, 8
                mul $r20 = $r20, 181 ;

        27270   sub $r10 = $r10, $r8
                mul $r11 = $r11, 3784
                sub $r5 = $r12, $r9 ;

        27271   sub $r10 = $r10, $r3
                add $r20 = $r20, 128
                brf $r26, label_232
```

in cui ciascuna istruzione lunga è identificata da un indice di esecuzione, un insieme comprendente da 1 a 4 operazioni e termina con un ";".

In questo esempio si può osservare un confine che conclude un blocco basico in corrispondenza dell'istruzione 27271 (l'operazione di salto condizionale).

Se si considera il tempo di vita dell'assegnazione del registro $r18 in 27268 ($D_n$) si vede che questa definizione viene utilizzata per l'ultima volta in 27269, dal momento che c'è un'altra definizione del registro $r18 nello stesso ciclo (vale a dire, D_{n+1}). Il valore di L_n di $r18 è pertanto pari ad un ciclo di clock. Va notato che non è possibile calcolare il valore L_{n+1} di $r18 dal momento che non ci sono né ultimi utilizzi U_{n+1} né ridefinizioni D_{n+2} nello stesso blocco basico.

Ai fini dell'analisi è stato considerato un insieme di programmi di prova (benchmark set) costituito dai seguenti algoritmi:

- un filtro con risposta all'impulso finita (FIR),
- un programma campione che realizza una DCT (Discrete Cosine Transform) ed un IDCT (Inverse Discrete Cosine Transform),
- una DCT ottimizzata,

- una IDCT ottimizzata, e
- una trasformata "wavelet".

Si noti che le versioni ottimizzate degli algoritmi DCT/IDCT sono caratterizzate, per migliorare le prestazioni, da un numero più basso di accessi a memoria e da un più elevato ri-utilizzo dei registri rispetto agli altri algoritmi.

La distribuzione dei valori di tempo di vita dei registri rilevati dagli algoritmi presi in considerazione è riportata nella tabella 1 che segue ed è sintetizzata in modo grafico nella figura 1 dei disegni annessi.

Durata di vita dei registri (cicli di clock)

Algoritmo	1	2	3	4	5	6	7	8
FIR	0%	13%	10%	10%	0%	0%	0%	0%
DCT/IDCT	28%	12%	8%	3%	2%	1%	1%	0%
DCT(ot.)	32%	14%	11%	6%	2%	1%	0%	0%
IDCT (ot.)	42%	12%	6%	5%	2%	1%	1%	1%
Wavelet	7%	17%	1%	0%	2%	0%	0%	0%

Nella tabella le colonne rappresentano la percentuale di registri la cui vita è pari ad un valore L dato compreso nel campo da 1 a 8 cicli di clock (istruzioni).

Nella figura 1 dei disegni annessi, la scala delle ordinate rappresenta il suddetto valore di percentuale in funzione del valore di L, riportato sulla scala delle ascisse.

Tanto la tabella quanto la figura 1 consentono di rendersi conto del fatto che - pur con ipotesi semplificative - per gli algoritmi ottimizzati si riscontra che circa la metà di tutte le definizioni di registro presenta valori di tempo di vita non superiori

a 2 cicli di clock (rispettivamente il 46% e il 54% per l'algoritmo DCT e l'algoritmo IDCT). In media, nel 35,4% dei casi la distanza fra la definizione del registro e l'ultimo suo impiego è minore o uguale a due cicli di clock, mentre nel 42,6% la distanza è minore o uguale a 3 cicli di clock.

L'analisi non tiene inoltre conto del caso in cui un registro non venga mai letto fra due definizioni successive. In realtà si può avere una sovrascrittura del registro, ad esempio, attraverso blocchi basici o durante commutazioni di contesto del processore (in risposta ad un'interruzione esterna, per esempio), ma tale fenomeno non può essere stimato in modo statico all'atto della compilazione nell'ambito di un blocco basico. Seppur vantaggioso per la soluzione secondo l'invenzione, il fenomeno è comunque non rilevante per l'analisi corrente, la quale pone l'accento su una funzione di ottimizzazione applicabile nell'ambito di un blocco basico durante la fase di compilazione statica VLIW.

Descrizione particolareggiata di un esempio preferito di attuazione dell'invenzione

Lo schema della figura 2 si riferisce, a titolo di esempio non limitativo, ad un'architettura di processore VLIW su quattro vie con una pipeline su cinque stadi provvista di logica di inoltro.

Gli stadi della pipeline sono i seguenti:

- IF: prelievo delle istruzioni (Instruction Fetch) dalla cache istruzioni (I-cache),
- ID: decodifica delle istruzioni e lettura degli operandi dal banco di registri RF,
- EX: esecuzione delle istruzioni in unità aritmetico logiche (ALU) con latenza pari ad un ciclo,
- MEM: accessi a memoria per istruzioni di caricamento/memorizzazione,

- WB: ri-scrittura degli operandi nel banco di registri RF.

Tre cammini di inoltro (EX-EX, MEM-EX e MEM-ID) forniscono collegamenti diretti fra coppie di stadi attraverso i registri interstadio EX/MEM e MEM/WB.

I vari simboli e designazioni riportati nella figura 2 sono ben noti ai tecnici esperti del settore e non richiedono pertanto di essere descritti in dettaglio in questa sede. Ciò vale tanto in relazione al loro significato, quanto riguardo alla loro funzione.

L'architettura in questione è applicabile, ad esempio, nei core VLIW di tipo embedded della famiglia Lx sviluppata congiuntamente dagli Hewlett-Packard Laboratories e dalla Richiedente. Ciascun cluster della famiglia Lx comprende quattro unità aritmetico logiche (ALU) per operandi interi su 32 bit, due moltiplicatori 16x32 ed un'unità di caricamento/memorizzazione. Il banco di registri RF comprende 64 registri general purpose su 32 bit e 8 registri di branching su un bit.

Con riferimento alla rete di inoltro considerata in precedenza, si consideri una sequenza $W = w_1 \ldots w_2 \ldots w_n$ di istruzioni lunghe. Una generica istruzione w_k può leggere i suoi operandi prodotti dalle seguenti istruzioni:

- w_{k-1} attraverso il cammino di inoltro EX/EX (utilizzato quando w_k è nello stadio EX),
- w_{k-2} attraverso il cammino di inoltro MEM/EX (utilizzato quando w_k è nello stadio EX),
- w_{k-3} attraverso il cammino di inoltro MEM/ID (utilizzato quando w_k è nello stadio ID),
- w_{k-n} quando $n > 3$ nel banco di registri RF.

Come indicato, la soluzione secondo l'invenzione inibisce la scrittura e le successive letture degli operandi nel banco di registri RF ogni volta che i

valori scritti possono essere rintracciati a partire dalla rete di inoltro per il loro ridotto tempo di vita.

Ciò avviene in modo specifico attraverso il segnale *Write inhibit* generato selettivamente nello stadio ID e destinato ad agire su un nodo WI interposto nel cammino del segnale di riscrittura *Write back* dallo stadio WB verso il banco di registri RF.

Ipotizzando, ad esempio, che un'istruzione w_d assegni un registro R il cui tempo di vita è minore o uguale a 3 e che w_k utilizzi R durante questo intervallo di vita, l'idea di base è quella di ridurre l'assorbimento di potenza:

- disabilitando la scrittura di R nello stadio WB di w_d, e
- impedendo a w_k di asserire l'indirizzo per la lettura dal banco di registri RF per leggere R (ottenuto a partire dalla rete di bypass).

In generale, mentre il fatto di evitare la riscrittura deve essere esplicitamente indicato nell'istruzione lunga w_d, l'informazione relativa al fatto che gli operandi sorgente debbono essere derivati dai cammini di inoltro viene in ogni caso resa disponibile dalla logica di controllo, quale che sia il tempo di vita della variabile. Di conseguenza è possibile evitare la lettura dal banco di registri RF ogniqualvolta ci si attende che gli operandi sorgente vengano estratti a partire dai cammini di inoltro (bypass).

La funzione di ottimizzazione dell'assorbimento di potenza descritta viene implementata con una logica dedicata nello stadio ID che disabilita i segnali di abilitazione in scrittura nel banco di registri RF e minimizza l'attività di commutazione delle porte di lettura del banco di registri RF mantenendo i gli

indirizzi di lettura in ingresso pari a quelli dell'ultimo ciclo di accesso utile.

Come esempio pratico, si può far riferimento alla sequenza di istruzioni considerate in precedenza ed in particolare alle istruzioni denominate 27268 e 27269. La riscrittura dei registri $r18, $r17 e $r19 nel banco di registri RF durante l'esecuzione di 27268 può essere evitata, e la successiva lettura di tali valori durante l'esecuzione di 27269 può essere effettuata direttamente a partire dal cammino EX-EX della rete di bypass.

In un processore superscalare, questo comportamento dovrebbe essere controllato tramite hardware, analizzando la finestra di istruzione così da calcolare il tempo di vita di un registro e generare segnali di controllo verso gli stadi della pipeline.

In un'architettura VLIW, tutte le decisioni di schedulazione che riguardano i dati, le risorse ed il controllo vengono risolte durante la compilazione in fase di schedulazione del codice, così come descritto, ad esempio nel lavoro di A. V. Aho, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, and Tools," Addison-Wesley, 1986.

Di conseguenza, la decisione relativa al fatto se il registro di destinazione debba essere inibito per quanto riguarda la scrittura oppure no, può essere demandata al compilatore, mantenendo limitato l'overhead di hardware.

Per passare l'informazione dal compilatore alla logica di controllo hardware è possibile seguire due vie diverse:

- riservare specifici bit di operazione in fase di codifica del formato di un'istruzione lunga: soluzione adatta in fase di definizione del set di istruzioni,

sebbene suscettibile di aumentare leggermente la lunghezza di codifica delle istruzioni; o

- sfruttare i bit di codifica delle istruzioni non utilizzati: soluzione attuabile quando il set di istruzioni è già stato definito, con possibilità di risparmiare lunghezza di istruzioni ma con l'eventualità di limitare le economie in termini di potenza ad un sottoinsieme delle operazioni presenti nel set di istruzioni.

In entrambi i casi, minimizzando l'attività di commutazione del banco di registri RF, si aumenta leggermente l'attività di commutazione delle unità di memoria in cui si immagazzinano le istruzioni.

Per quanto riguarda il problema della gestione delle eccezioni, si può ipotizzare che lo stato del processore possa essere:

- uno stato di architettura permanente immagazzinato nel banco di registri RF, o
- uno stato di architettura volatile immagazzinato nei registri interstadio della pipeline a partire dai quali la rete di inoltro trasferisce gli operandi sorgente.

Lo stato di architettura volatile viene trattato come una memoria FIFO (First In First Out), la cui capacità è pari al numero di stadi durante i quali il risultato di un'operazione può essere immagazzinato nella pipeline (nel caso dell'architettura su cinque stadi rappresentata nella figura 2, tale capacità è pari a tre).

In generale, un processore pipeline assicura che quando un elemento esce dallo stato volatile, esso viene automaticamente riscritto nel banco di RF.

Al contrario, nella soluzione qui descritta, quando un elemento esce dallo stato volatile e non

viene più utilizzato, esso può essere scartato evitando la riscrittura del banco di RF.

Questo comportamento può creare alcun problemi quando si manifesta un'eccezione durante l'elaborazione.

Nell'architettura qui trattata come esempio di riferimento, un'eccezione può presentarsi in particolare durante gli stadi ID, EX o MEM e può essere trattata nello stadio WB.

Sulla base della tassonomia delle eccezioni definita nel lavoro H. Corporaal già citato in precedenza, si assume che il processore adotti il modo di funzionamento correntemente denominato "user-recoverable precise mode".

Sulla base di questo modello, le eccezioni possono essere esatte o inesatte.

Un'eccezione esatta causata da un'istruzione emessa al tempo t è un'eccezione precisa tale da richiedere che i cambiamenti di stato causati dal trattamento delle eccezioni risultino visibili per tutte le istruzioni emesse al e dopo l'istante t ed a nessuna delle istruzioni emesse in precedenza. Inoltre, tutti i cambiamenti di stato delle istruzioni emesse prima dell'istante t sono visibili alla funzione di trattamento dell'eccezione.

Se si assume che le eccezioni siano trattate nel modo esatto, quando l'istruzione costituente un'eccezione raggiunge lo stadio WB, le istruzioni sulla pipeline vengono eliminate (flushed) e rieseguite.

Si faccia riferimento alla situazione della figura 3, al ciclo x un'istruzione w_k legge i suoi valori da un'istruzione w_{k-2} inibita per quanto riguarda la scrittura attraverso la rete di inoltro. Allo stesso tempo, si supponga che l'istruzione w_{k-1} generi

un'eccezione durante lo stadio MEM. I risultati di w_{k-2} sarebbero perduti, ma è invece necessario che questi valori vengano utilizzati durante la riesecuzione di w_k. Dal momento che né la rete di inoltro né il banco di RF contengono i risultati di w_{k-2}, lo stato di architettura visto durante la riesecuzione di w_k (al ciclo x+nn) risulterebbe non corretto.

Al fine di garantire che le istruzioni nella pipeline vengano rieseguite nello stato corretto del processore, i valori inibiti per quanto riguarda la scrittura devono essere scritti nel banco di registri RF ogniqualvolta negli stadi ID, EX o MEM si genera un segnale di eccezione.

Nel caso dell'esempio precedente, vale a dire con w_{k-1} che genera un'eccezione nello stadio MEM, la soluzione qui descritta forza la riscrittura dei risultati di w_{k-1} e w_{k-2} nel banco di RF, per cui durante la riesecuzione di w_k al ciclo x+nn gli operandi vengono letti dal banco di RF.

Qualora si assuma invece che le eccezioni vengano trattate nel modo non esatto o 'inesatto', quando si ha un'eccezione, le istruzioni presenti nella pipeline vengono eseguite fino al completamento, senza osservare gli effetti dell'eccezione che viene trattata successivamente. In questo caso, tutte le istruzioni nella pipeline vengono costrette a riscrivere i risultati nel banco di registri.

L'architettura rappresentata in figura 2 è in grado di garantire entrambi i meccanismi di gestione delle eccezioni descritti in precedenza.

Quando le eccezioni vengono trattate in modo esatto, il tempo di vita di un registro supportato è minore o uguale a due cicli di clock (attraverso i cammini EX/EX e MEM/EX).

Quando le eccezioni vengono gestite in modo non esatto, il tempo di vita del registro che le utilizza può essere esteso a tre cicli di clock (attraverso i cammini EX/EX, MEM/EX e MEM/ID).

Per quanto riguarda specificatamente il caso degli interrupt o i fenomeni di "cache miss", la natura asincrona degli interrupt consente di trattarli come eccezioni non esatte forzando ciascuna istruzione lunga nella pipeline a riscrivere i risultati prima di trattare l'interrupt; i fenomeni di cache miss producono invece fenomeni assimilabili a bolle che si propagano lungo la pipeline, per cui ogni volta che la logica di controllo della cache solleva un segnale di miss, si forza la riscrittura dei risultati delle istruzioni nella pipeline.

Ad ulteriore chiarificazione della descrizione fornita in precedenza si può notare che, secondo uno degli elementi di maggiore interesse dell'invenzione, le sezioni dati dei registri interstadio della struttura a pipeline diventano in pratica un ulteriore strato - a livello più alto - della gerarchia di memoria.

Nel seguito questi registri verranno chiamati microregistri.

Essi sono visibili al compilatore ma non al programmatore.

Le regole di ottimizzazione per il loro impiego sono particolari, e diverse da quelle degli elementi del banco di registri.

Essi non sono indirizzabili in scrittura (o meglio, vengono indirizzati in modo implicito) e le regole per l'indirizzamento in lettura sono legate all'architettura, essendo più restrittive rispetto a quanto avviene per gli elementi del banco di registri.

Come si è indicato, la soluzione secondo l'invenzione si basa essenzialmente sulla funzione di inoltro (bypass) così da evitare le scritture e le letture nel banco di registri così da ridurre l'assorbimento di potenza.

Ogni volta che identifica variabili a vita breve tali da rendere possibile la soluzione di inoltro, ed avendo verificato che sono riscontrate le condizioni indicate nel seguito, il compilatore non riserva registri nel banco di registri per tali variabili.

Per quanto riguarda l'uso da parte del compilatore, lo spazio del banco di registri viene così di fatto aumentato, riducendo quindi il fenomeno di register spilling ed il traffico di cache che ne deriva.

Si consideri in dettaglio la struttura pipeline su cinque stadi schematicamente rappresentata nella figura 4 e complessivamente simile a quella rappresentata nella figura 2, con la previsione tuttavia di due stadi EX1 ed EX2.

Si prenda come esempio un'istruzione di linguaggio di alto livello del tipo:

x:=a*b+c-d

che viene tradotta in codice intermedio come:

t_0= a*b

t_1=c-d

x=t_0+t_1.

Si supponga che la latenza di operazione sia pari ad 1 per la sottrazione e a 2 per la moltiplicazione.

Se si indica con μ_0 la sezione di risultato nel latch all'uscita dello stadio EX2 e con μ_1 la corrispondente sezione nel latch all'uscita dello stadio EX1, le tre operazioni elementari viste in precedenza si traducono in un linguaggio

pseudoassemblatore che sfrutta i microregistri nel modo seguente

mul μ_2, R1, R2	si assume che a, b, c, d siano inizialmente immagazzinati nei registri R1 a R4
sub μ_1, R3, R4	
add R5, μ_1, μ_2	il risultato finale è immagazzinato in R5.

con i cammini di inoltro a partire dai latch che vengono sfruttati così come rappresentato appunto nella figura 4.

Per una pipeline su cinque stadi, la distanza massima ammissibile fra la scrittura di una variabile in un microregistro e l'impiego della stessa variabile è pari a tre. Ciò crea una limitazione per il compilatore, che è in grado di sfruttare i microregistri solo nella misura in cui è possibile una schedulazione nell'ambito della distanza accettabile. Naturalmente, se si utilizzano pipeline più profonde è possibile utilizzare distanze più elevate (e procedure di schedulazione più complesse, nonché un'ulteriore riduzione nell'impiego del banco di registri RF).

Questo primo esempio si riferisce ad un codice sequenziale. Nel caso di cicli (loop) i microregistri possono essere utilizzati anche attraverso i confini dei cicli, purché le limitazioni indicate in precedenza risultino soddisfatte tanto fra i cicli quanto all'interno dei cicli.

Se si considera l'estensione ad una semplice (pura) architettura VLIW, il punto interessante è costituito dalla possibilità di avere sillabe (in parallelo in un'unica istruzione lunga) caratterizzate da latenze diverse.

In questo caso può essere necessario tenere conto dei trasferimenti fra microregistri lungo i cammini della pipeline.

Si consideri ancora lo stesso segmento di codice visto in precedenza, ed una struttura VLIW con due cammini (lane) con un'unità aritmetico-logica ed un moltiplicatore, ossia una struttura corrispondente a quella rappresentata nella figura 5.

Si supponga inoltre che le latenze siano le stesse viste in precedenza. Il codice viene pertanto schedulato nel modo seguente

i1 mul μ_0^2, R1, R2; sub μ_1^1, R3, R4
l'apice indica lo stadio, il pedice il percorso

i2 nop
il contenuto di μ_1^1 scorre lungo il percorso verso μ_1^2, mentre il risultato finale della moltiplicazione è immagazzinato in μ_0^2

i3 add R5, μ_0^2, μ_1^2.

Inoltre, se i cammini di inoltro presenti nella microarchitettura lo consentono, si possono prevedere trasferimenti dai microregistri in un percorso verso unità funzionali in un diverso percorso. In ogni caso, le limitazioni base per il compilatore - oltre a quelle inerenti alla latenza - sono le seguenti:

- il microregistro di scrittura è sempre quello nel percorso dove si trova l'unità funzionale; i trasferimenti in avanti lungo la pipeline sono quindi limitati allo stesso percorso;

- la lettura dai microregistri è sempre permessa nell'ambito dello stesso percorso e - quando si ha a che fare con diversi percorsi - nella misura in cui i percorsi di bypass lo permettono.

L'impiego dei microregistri può creare criticità quando si hanno da trattare interrupt (e, più in generale, eccezioni).

In effetti, i microregistri possono essere visti come tali da costituire una memoria "transitoria", e dunque tali da non poter essere associati con uno stato della macchina da salvare nel caso di un'eccezione (tranne nel caso in cui si preveda una soluzione quale una pipeline "ombra").

Per quanto riguarda la gestione degli interrupt, si possono proporre due diverse possibili soluzioni a questo problema.

Una prima soluzione si basa sulla definizione di una 'sequenza atomica', nel senso che la sequenza di istruzioni che utilizzano i microregistri viene vista come una sequenza atomica e, come tale, non suscettibile di essere interrotta. L'interrupt viene disabilitato prima dell'inizio della sequenza e lo stato della macchina viene reso stabile (tramite scrittura nel banco di registri o nella memoria) prima che l'interrupt venga riabilitato. Questa soluzione non richiede un'estensione dell'insieme di istruzioni o della microarchitettura, e viene di fatto trattata soltanto dal compilatore.

Un'altra soluzione è basata su un principio suscettibile di essere denominato funzione di "checkpoint".

Vengono introdotte due nuove istruzioni (in realtà, pseudo-istruzioni utilizzate dal compilatore e che influenzano soltanto l'unità di controllo, non le pipeline), ossia una dichiarazione di checkpoint (ckp.d) e un rilascio di checkpoint (ckp.r).

Al momento della dichiarazione di checkpoint, il Contatore di Programma (PC) viene salvato in un registro ombra e fino al momento di rilascio del

checkpoint lo stato di macchina non può essere modificato (ovviamente, questo implica che non vengono consentite istruzioni di memorizzazione). Al momento del rilascio del checkpoint, il registro ombra viene resettato e gli interrupt vengono disabilitati in modo atomico. I risultati calcolati nella sezione sottoposta a checkpoint possono essere finalmente memorizzati (committed) modificando lo stato reale del processore e, dopo di questo, gli interrupt vengono nuovamente abilitati per far ripartire l'esecuzione normale. Nel caso di un interrupt fra ckp.d e ckp.r, il PC a partire dal quale riparte l'esecuzione dopo il trattamento dell'interrupt è quello salvato nel registro ombra (e, evidentemente, alla luce delle limitazioni sull'aggiornamento dello stato della macchina, lo stato della macchina è consistente con tale PC).

Al riguardo possono essere proposte due soluzioni alternative.

In base alla prima soluzione, tutte le scritture nei registri nella sequenza fra ckp.d e ckp.r coinvolgono soltanto dei microregistri. Il compilatore verifica se c'è una schedulazione tale da soddisfare tale limitazione. Il banco di RF coinvolto soltanto per leggere dati.

In base alla seconda soluzione, un (piccolo) sottoinsieme del banco di registri viene riservato in modo convenzionale per quelle variabili "transitorie" fra la dichiarazione del checkpoint ed il relativo rilascio, la cui vita utile supera quella massima consentita dalla lunghezza della pipeline. La prima comparsa di registri "transitori" nella sequenza sottoposta a checkpoint deve essere una definizione (o un caricamento o una scrittura verso un registro). Questi registri transitori non sono visti come una parte costituente dello stato della macchina dopo il

rilascio del checkpoint (ossia sono considerati valori morti dopo questo punto). Va notato che, ovviamente, l'adozione di questi registri transitori può implicare il rischio di un fenomeno di register spilling. Molto semplicemente, qualora dovesse risultare necessario procedere ad un register spilling, l'uso dei microregistri viene escluso e viene adottata la compilazione normale con l'impiego del banco di RF.

Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno essere ampiamente variati rispetto a quanto descritto ed illustrato, senza per questo uscire dall'ambito della presente invenzione, così come definita dalle rivendicazioni annesse.

RIVENDICAZIONI

1. Architettura di circuito processore di tipo pipeline comprendente una pluralità di stadi (IF, ID, EX, MEM, WB) ed una rete di cammini di inoltro (EX-EX, MEM-EX, MEM-ID) che collegano detti stadi nonché un banco di registri (RF) per la riscrittura degli operandi, caratterizzata dal fatto che comprende una funzione di ottimizzazione dell'assorbimento di potenza tramite inibizione (*Write inhibit*) della scrittura e delle successive letture in detto banco di registri (RF) di operandi rintracciabili a partire da detta rete di inoltro per il loro ridotto tempo di vita.

2. Architettura secondo la rivendicazione 1, caratterizza dal fatto che detta funzione è configurata per realizzare selettivamente, in relazione ad almeno un registro dato (R) assegnato da una prima istruzione (w_d) comprendente uno stadio di riscrittura (WB) ed utilizzato da una seconda istruzione (w_k):

- la disabilitazione della riscrittura di detto registro (R) in detto banco di registri (RF) nello stadio di riscrittura (WB) di detta prima istruzione, e
- l'inibizione dell'asserzione dell'indirizzo di lettura di detto registro (R) in detto banco di registri (RF) da parte di detta seconda istruzione (w_k).

3. Architettura secondo la rivendicazione 1 o la rivendicazione 2, caratterizzata dal fatto che comprende una logica dedicata per disabilitare i segnali di abilitazione di scrittura in detto banco di registri (RF).

4. Architettura secondo una qualsiasi delle precedenti rivendicazioni, caratterizzata dal fatto che comprende una logica dedicata che minimizza l'attività di commutazione delle porte di lettura in detto banco di registri (RF) mantenendo i valori sugli indirizzi di

lettura in ingresso in corrispondenza dei cicli di clock precedenti

5. Architettura secondo la rivendicazione 3 o la rivendicazione 4, caratterizzata dal fatto che comprende uno stadio (ID) di decodifica delle istruzioni e di lettura degli operandi da detto banco di registri (RF) e dal fatto che detta logica dedicata è compresa in detto stadio (ID).

6. Architettura secondo una qualsiasi delle precedenti rivendicazioni, caratterizzata dal fatto che detto processore è un processore superscalare comprendente un'unità controllo hardware suscettibile di analizzare la finestra di istruzione per determinare il tempo di vita dei registri.

7. Architettura secondo una qualsiasi delle rivendicazioni 1 a 5, caratterizzata dal fatto di essere configurata come architettura VLIW, in cui la decisione di attivare detta funzione di inibizione viene demandata al compilatore.

8. Architettura secondo la rivendicazione 7, caratterizzata dal fatto che il compilatore trasferisce l'informazione verso la logica di controllo hardware riservando specifici bit di operazione nella codifica delle istruzioni.

9. Architettura secondo la rivendicazione 7, caratterizzata dal fatto che il compilatore trasferisce l'informazione verso la logica di controllo hardware sfruttando bit di codifica delle istruzioni non utilizzati.

10. Architettura secondo una qualsiasi delle precedenti rivendicazioni, caratterizzata dal fatto che comprende registri interstadio (EX/MEM, MEM/WB) compresi fra gli stadi della struttura pipeline per immagazzinare uno stato di architettura volatile e dal fatto che l'architettura è configurata per scartare

degli elementi che escono da detto stato volatile, evitandone la riscrittura in detto banco di registri (RF).

11. Architettura secondo la rivendicazione 10, caratterizzata dal fatto di essere suscettibile di operare su istruzioni configurabili come eccezioni e dal fatto che, per assicurare la riesecuzione di istruzioni costituenti eccezione nello stato corretto del processore, è prevista la riscrittura dei valori inibiti in relazione alla scrittura in detto banco di registri (RF) in presenza di un segnale che si configuri come eccezione.

12. Architettura secondo la rivendicazione 11, caratterizzata dal fatto che comprende uno stadio (ID) di decodifica delle istruzioni e di lettura degli operandi da detto banco di dei registri (RF), uno stadio di esecuzione delle istruzioni (EX), ed uno stadio degli accessi di memoria (MEM) e dal fatto che detta riscrittura è prevista ogniqualvolta si genera un segnale di eccezione in uno di detti stadi (ID, EX, MEM).

13. Architettura secondo la rivendicazione 11, caratterizzata dal fatto che, in presenza di un'istruzione che si configuri come un'eccezione, l'architettura è configurata per eseguire fino al completamento le istruzioni sulla pipeline, prevedendo la riscrittura in detto banco di registri (RF) dei risultati di tutte le istruzioni sulla pipeline.

14. Architettura secondo una qualsiasi delle precedenti rivendicazioni, caratterizzata dal fatto che comprende registri, quali ad esempio registri latch, interstadio (μ_0, μ_1, μ_2) utilizzati come strato di memoria per la memorizzazione degli operandi.

15. Architettura secondo la rivendicazione 14, caratterizzata dal fatto che detti registri interstadio

(μ_0, μ_1, μ_2), sono configurati così da risultare visibili al compilatore e non al programmatore.

16. Architettura secondo la rivendicazione 14 o la rivendicazione 15, caratterizzata dal fatto che detti registri interstadio (μ_0, μ_1, μ_2) non sono indirizzabili in scrittura, essendo indirizzati in modo implicito.

17. Architettura secondo una qualsiasi delle rivendicazioni 14 a 16, caratterizzata dal fatto che detti registri interstadio (μ_0, μ_1, μ_2) sono configurati come memoria transitoria non associabile ad uno stato di macchina suscettibile di essere salvato nel caso di un'eccezione.

18. Architettura secondo la rivendicazione 17, caratterizzata dal fatto di essere configurata in modo tale per cui le sequenze di istruzioni che utilizzano detti registri interstadio (μ_0, μ_1, μ_2) vengono trattate come sequenze atomiche non suscettibili di essere sottoposte ad interrupt.

19. Architettura secondo la rivendicazione 18, caratterizzata dal fatto che è prevista la disabilitazione di qualunque interrupt prima dell'avvio di dette sequenze e dal fatto che lo stato della macchina viene reso stabile prima della riabilitazione dell'interrupt, ad esempio tramite riscrittura nel banco di registri o in memoria.

20. Architettura secondo la rivendicazione 17, caratterizzata dal fatto che comprende una funzione di generazione di due pseudo-istruzioni di dichiarazione e rilascio di checkpoint (ckp.d, ckp.r) rispettivamente con la previsione di un registro ombra cui il contatore di programma (PC) viene salvato dal momento di dichiarazione del checkpoint, lo stato di macchina non potendo essere modificato fino al rilascio del checkpoint, per cui al rilascio del checkpoint, il

registro ombra viene ripristinato e gli interrupt disabilitati in modo atomico.

21. Architettura secondo la rivendicazione 20, caratterizzata dal fatto che i risultati calcolati fra dette due pseudo-istruzioni vengono affidati allo stato reale del processore con successiva riabilitazione degli interrupt per consentire il riavvio dell'esecuzione normale.

22. Architettura secondo la rivendicazione 20 o la rivendicazione 21, caratterizzata dal fatto che, in presenza di interrupt fra dette pseudo-istruzioni, l'esecuzione viene fatta ripartire dopo il trattamento degli interrupt a partire dal contatore di programma (PC) immagazzinato nel registro ombra.

23. Architettura secondo la rivendicazione 20, caratterizzata dal fatto che tutte le scritture di registri comprese fra dette pseudo-istruzioni coinvolgono soltanto detti registri interstadio, per cui detto banco di registri (RF) viene coinvolto solo per la lettura dei dati.

24. Architettura secondo la rivendicazione 20, caratterizzata dal fatto che comprende un sottoinsieme del banco di registri (RF) riservato per le variabili transitorie generate fra dette due pseudo-istruzioni e la cui vita eccede il valore massimo concesso dalla pipeline.

25. Architettura secondo la rivendicazione 24, caratterizzata dal fatto che la prima apparizione di registri transitori nella sequenza sottoposta a checkpoint è una definizione quale un caricamento o una scrittura in un registro, suscettibile di essere vista come parte costitutiva dello stato di macchina dopo il rilascio del checkpoint.

RIASSUNTO

Un'architettura di circuito processore di tipo pipeline, preferibilmente del tipo VLIW, comprende una pluralità di stadi (IF, ID, EX, MEM, WB) ed una rete di cammini di inoltro (EX-EX, MEM-EX, MEM-ID) che collegano coppie di detti stadi nonché un banco di dei registri (RF) per la riscrittura degli operandi. E' prevista una funzione di ottimizzazione dell'assorbimento di potenza tramite inibizione (*Write inhibit*) della scrittura e delle successive letture in detto banco di dei registri (FR) di operandi rintracciabili a partire da detta rete di inoltro per il loro ridotto tempo di vita.

(Figura 2)

Fig. 1

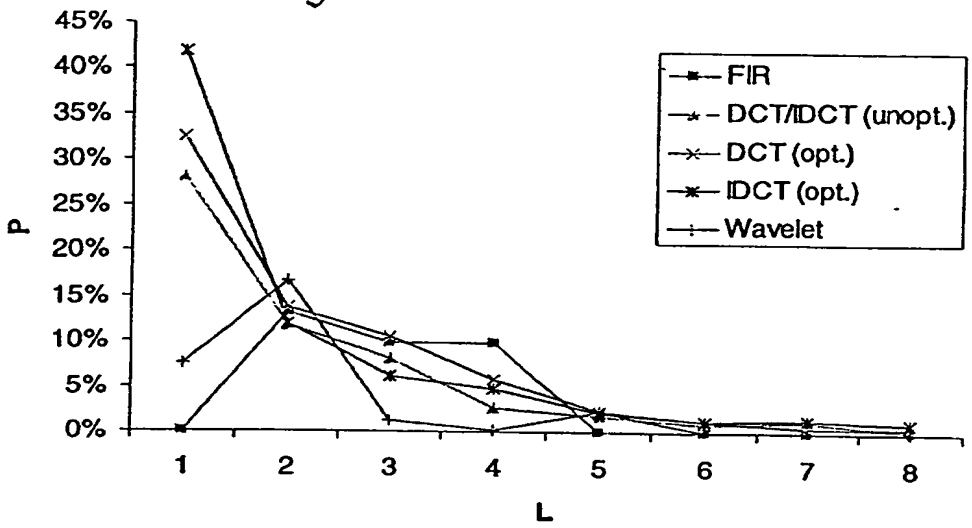
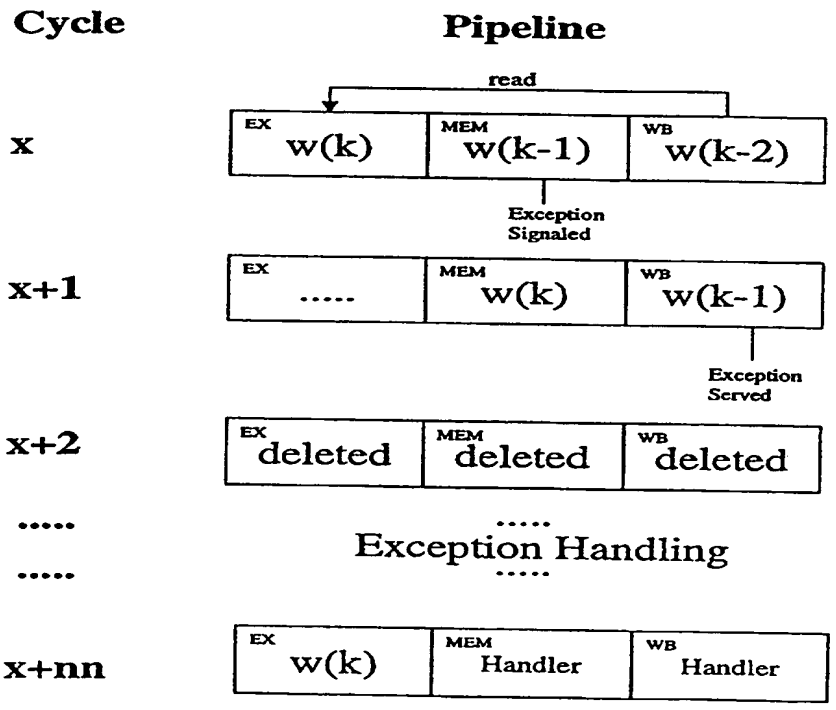

45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
P
1
2
3
4
5
6
7
8
L
FIR
DCT/IDCT (unopt.)
DCT (opt.)
IDCT (opt.)
Wavelet

Fig. 3


Cycle
Pipeline
read
x
EX w(k)
MEM w(k-1)
WB w(k-2)
Exception Signaled
x+1
EX .....
MEM w(k)
WB w(k-1)
Exception Served
x+2
EX deleted
MEM deleted
WB deleted
.....
Exception Handling
.....
x+nn
EX w(k)
MEM Handler
WB Handler

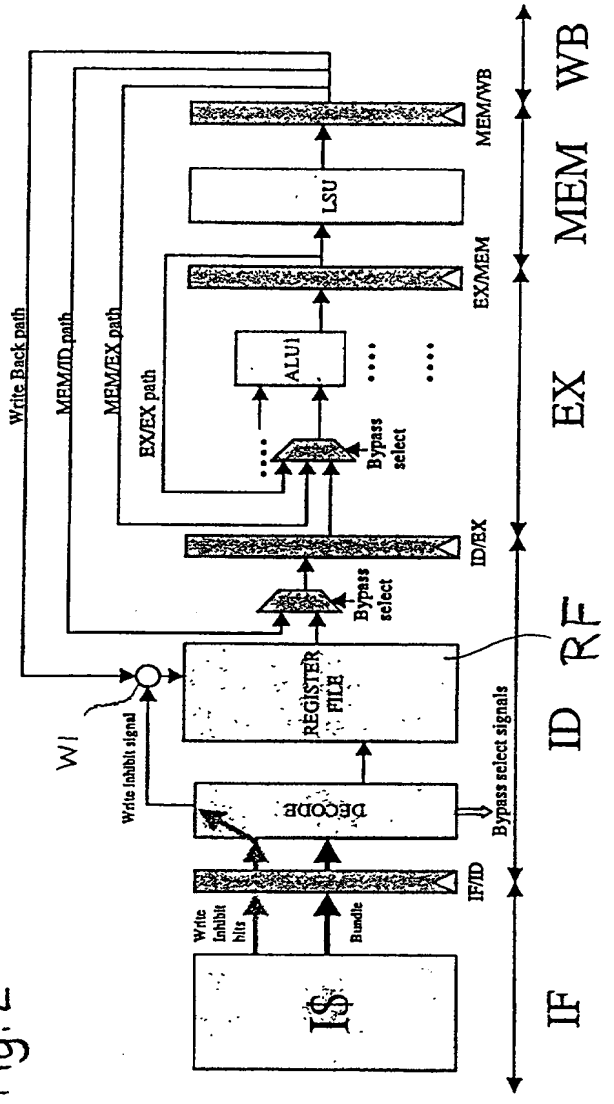
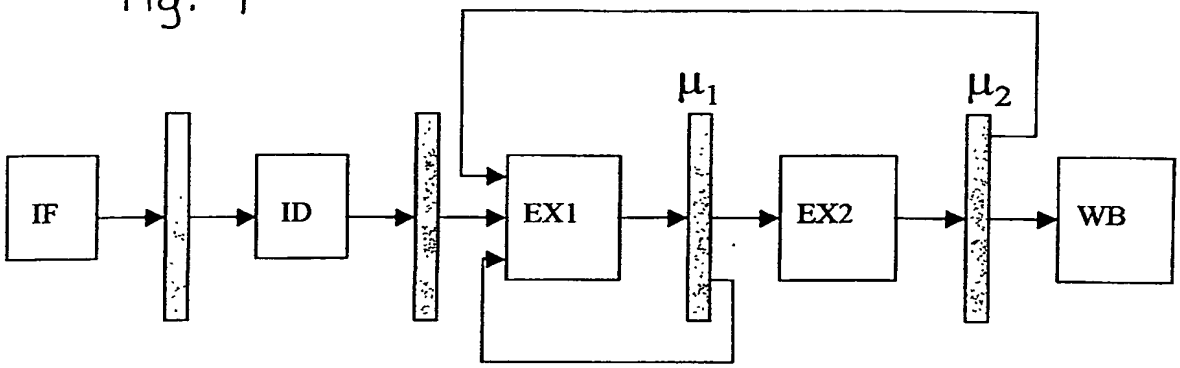

IF
ID RF
EX
MEM
WB
I$
Write inhibit bits
Bundle
IF/ID
Decode
Write inhibit signal
WI
Bypass select signals
REGISTER FILE
Bypass select
ID/EX
ALU1
LSU
EX/MEM
MEM/WB
Write Back path
MEM/ID path
MEM/EX path
EX/EX path

Fig. 2

Fig. 4



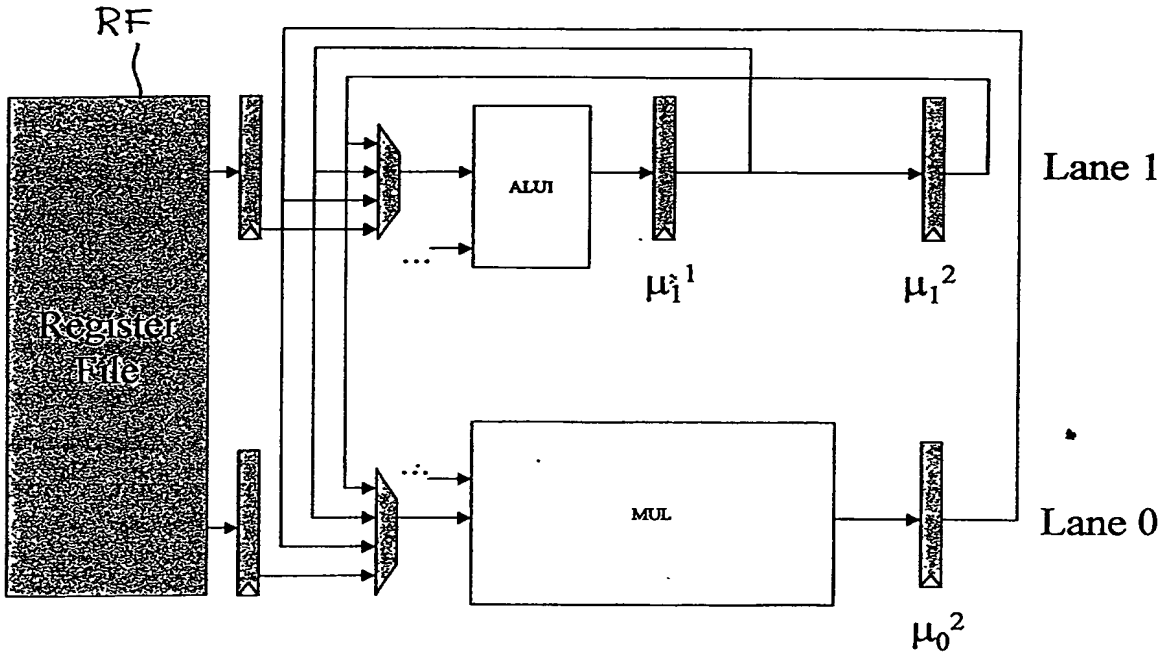


Fig. 5